ICIC Express Letters
Volume **18**, Number **2**, February **2024**

 ISSN 1881-803X
pp. 115–123

PERFORMANCE EVALUATION ON MULTITHREADED EIGENFACE PROGRAM USING A SINGLE BOARD COMPUTER

Adnan[1,*], Asrarul Ikram[2] and Syafaruddin[3]

[1]Department of Informatics
[3]Department of Electrical Engineering
Universitas Hasanuddin
Jl. Poros Malino Km. 6 Bontomarannu, Gowa 92127, Sulawesi Selatan, Indonesia
syafaruddin@unhas.ac.id
*Corresponding author: adnan@unhas.ac.id

[2]Department of Technology and Engineering
PT. Yuscorp Bisnis Indonesia
Jl. Perintis Kemerdekaan (Puri Asri Raya Blok AE/5), Tamalanrea 90245, Indonesia
hello.asrarulikram@gmail.com

Received March 2023; accepted June 2023

Abstract. *This paper aims to assess how a single-board computer such as BPI M3 can handle a parallel program and enhance its performance. BPI M3 can perform a face recognition program written with OpenMP, enhancing its response time. The program encompasses several functions, such as computing the covariance matrix, calculating Eigenface, and calculating the weight. The program performs Face recognition on several different image sizes, namely 92 × 112, 296 × 360, 394 × 480, and 493 × 600. While the accuracy of facial recognition remains the same, parallel programming improves the program response time. As a result, we found the best speedup is 2.35 on eight cores of BPI M3. This result is fair due to overheads and small of parallel fractions in the program.*
Keywords: Eigenface, Multithreaded, Single-board computer, Speedup, OpenMP

1. **Introduction.** As long as a program has low time complexity and input size, serial programming is adequate. However, if one or both are high, then serial programming will lead to slow response times. Therefore, parallel programming is needed to speed up the computation [1, 2, 3]. With parallel programming, programmers need to understand the complexity and cases associated with data and task dependencies [4, 5, 6]. A multiprocessor contains more than one such CPU (Central Processing Unit) and allows them to work in parallel. There are two or more CPUs integrated into a single computer system. So basically, it has two or more CPUs on the system physically. Simultaneously the processor performs different program parts, thus speeding up the computation process. As the number of transistors doubled every 18 months [7], computer architects decided to duplicate more CPUs and integrate the CPUs with a cache memory hierarchy into a chip. This new approach is known as multicore [8]. Each core is a complete architecture capable of running processes or threads. Multicore is sometimes called CMP, Chip-level Multiprocessing [9, 10, 11].

The discovery of multicore CPUs was not without purpose. The design and implementation of multicore CPUs are intended to improve performance while reducing electricity consumption [12]. To achieve the goal of improving performance while also increasing electrical energy consumption efficiency, multicore CPUs must be utilized with optimized

DOI: 10.24507/icicel.18.02.115

program code [13]. The appropriate form of source code optimization is through multithreading [14, 15]. Therefore, rewriting sequential program source code into multithreaded code is the motivation for this research.

In this research, a face recognition program using the Eigenface algorithm is used as a case study. The selection of the Eigenface program, besides being a realistic benchmark, also presents challenges in achieving high performance.

From the perspective of using single-board computers, the utilization of low-power single-board computers in research has recently emerged as a trend. For example, they have been employed in the implementation of IoT in [16], in the deployment of low-cost network control systems in [17], healthcare application [18], and in person tracking in [19]. Enriching the body of knowledge regarding the use of low-power single-board computers, as well as testing them with a wider range of applications, needs to be done.

In recent years, there have been extensive studies on facial recognition, and various related fields have contributed to increasing accuracy in areas such as images and video [20, 21, 22]. The study in [20] provided performance evaluation results in terms of accuracy, but did not attempt to improve processing time. The study in [21] also presented performance results in terms of face recognition accuracy, but on chimpanzee faces, and did not improve processing time either. The study in [22] presented performance improvements using a heterogeneous system (CPU-GPU computation). While massively parallel processors can provide faster performance compared to CPUs, the use of hundreds to thousands of GPU cores does not provide linear speedup according to Amdahl's law equation. For example, the study only resulted in a speedup of 32 despite using the AMD Radeon GPU with at least a thousand cores. Communication between the CPU and GPU for each kernel launch will add processing time. Furthermore, the use of both CPU and GPU consumes more electrical resources, making it unsuitable for implementation on portable devices.

The bigger the input data, the longer the computation, thus attracting our attention to conduct this research. Adreeva has developed a face recognition program with an Eigenface algorithm [23]. The program development uses the C++ language in serial mode. Although the program helps to recognize faces, the program requires an improvement for better CPU utilization. Lindner et al. [24] have also tested various single-board computers, one of which is the Banana Pi M3. The authors evaluated the Face recognition program on the BPI M3, but the program is a serial program instead of a parallel program.

We emphasize the novel contributions of this paper as follows.

1) We have developed a multithreaded program for Eigenface to increase the speed of both the training and inference processes. To the best of our knowledge, none of the authors have applied multithreaded techniques in the Eigenface program for the single board computer before.
2) In order to point out the benefit of the multithreaded method, we present detailed performance evaluations of the multithreaded Eigenface program. To the best of our knowledge, none of the authors have reported the multithreaded Eigenface program's performance in a paper before.
3) We propose a new approach to analyzing the performance using Amdahl's law. The new approach involves using single-threaded parallel processing time instead of serial processing time.

In this paper, there are five sections. The first section is the introduction, and Section 2 discusses the problem to address. Section 3 describes the experimental setup. We present and discuss the results in Section 4. Finally, Section 5 contains conclusions.

2. **Problem Statement and Preliminaries.**

In a computer program there may be several parts that must be executed serially and maybe several parts that can be executed in parallel. The serial portion of the program is called the serial fraction while the parallel

portion is called the parallel fraction. The sum of both serial fractions and parallel fractions is defined in Equation (1), where s is the serial fractions and f is the parallel fraction.

$$s = 1 - f \tag{1}$$

Parallel programming can improve the performance of computer programs. However, parallel programming may not apply to all parts of the program. Parallel programming only applies to parallel fractions. According to Amdahl's law [25], a speedup variable, a quantitative metric for improving program performance as a result of parallel programming, is directly proportional to the magnitude of the parallel fraction. Equation (2) defines the speedup metric as a function of the parallel fraction magnitude, where P is the number of processors.

$$S(f, P) \leq \frac{1}{s + \frac{f}{P}} \tag{2}$$

According to the discussion mentioned above, the solution of parallel programming is to parallelize all parallel fractions. However, since there is no free lunch, parallelizing every parallel fraction is not always beneficial. Parallelizing a parallel fraction makes sense if the benefits outweigh the costs. Therefore, the research problem is which part of the Eigenface program is beneficial to parallelize.

Overhead is the cost incurred by parallel work. This overhead is due to the processors performing extra work, thereby increasing the total working time. The extra works include scheduling, context switching, synchronization, and communication. Idle time due to load imbalance is also considered overhead because it increases the total working time. Because the processor does no extra work on serial programs, there is no overhead. Therefore, the complete works of the parallel program are more than the complete works of the serial program, which is expressed by Equation (3). The definition of overhead is embedded in the T_P of the equation, where T_S is whole serial program's execution time, P is the number of processors, and T_P is the execution time of the parallel program.

$$T_S \leq P \times T_P \tag{3}$$

T_1 is the parallel program's execution on $P = 1$ processor. In T_1, only parallel runtime systems (parallelization APIs) contribute to overhead. Idle, synchronization and communication should not exist. Here, the additional works in T_1 is defined in Equation (4).

$$T_S < T_1 \tag{4}$$

As we add the number of processors, idle, synchronization, and communication contribute to overhead and execution time inflation, as Equation (5) defines.

$$T_1 \leq P \times T_P \tag{5}$$

Although T_1 is not very accurate for estimating speedup, estimation by T_1 is still more accurate than T_S. It is because T_1 has taken account of overhead.

3. **Method.** We propose a multithreading method to accelerate the Eigenface program in order to maximize the utilization of the multicore CPUs found on the Banana Pi M3. In a multithreaded process, multiple execution threads share work. In the case of a multithreaded Eigenface program, the work is a parallel iterative computation executed by multiple threads in parallel.

Compared to the most recent stage of development of Eigenface [22], which implements parallel Eigenface in the SIMD (Single Instruction Multiple Data) model, our implementation of parallel Eigenface follows a task parallelism approach. In the task parallelism model, multiple threads create tasks, add them to the task pool, and execute them simultaneously. This approach offers the advantage of better performance portability, meaning that it can scale with different numbers of CPUs on various machines.

To find the profitable parallel fraction of the program, in the experiment, we do profiling of the serial version of the program first. Generally, those few algorithm parts have the most significant time complexity. Specifically, in the iterative case, it shows a nested loop structure. For the case of the Eigenface program used in this research, we applied profiling to the eight C functions below the C main function. The eight functions are listed in Table 1. The C main function consumes execution time for memory allocation task for some data structures.

TABLE 1. The eight tasks (C functions) to which profiling is performed

No	Task	Task output	Remarks
1	read_training_data	A (image vectors)	Serial, IO Bound
2	mean_image	$\bar{A}$	Low complexity
3	substract mean image	$A = A - \bar{A}$	Low complexity
4	transpose_matrix	A^T	Low complexity
5	covariance_matrix	$S = A \times A^T$	High complexity
6	Jacobi Eigenvalue	V of S	Low complexity
7	Eigenface	$U = V \times A$	High complexity
8	Weight	$W = U \times A^T$	High complexity

After the experiment produces a serial program execution profile, we parallelize the program with OpenMP directives and APIs. The OpenMP directives and APIs annotate the serial program source. An OpenMP compiler translates the annotated source to the multithreaded code.

In this experiment, we parallelize only three tasks with the highest computational complexity: the covariance matrix, Eigenface, and weight. We use 1, 2, 4, and 8 threads. To account for overhead in scalability analysis using Amdahl's law, this paper compares scalability based on T_S/T_P, T_1/T_P, and estimations.

3.1. **Experimental setting.** In this work, we rewrite the Eigenface program on a single-board computer called Banana Pi M3. Banana Pi is a low-cost and credit card-sized single-board computer manufactured by the Shenzhen SINOVOIP Co., Ltd. Its spin-off Guangdong BiPai Technology Co., Ltd., NetBSD, Android, Ubuntu, Debian, and Arch Linux can run on Banana Pi. It uses the Allwinner SoC with a 1.8GHz octa-core. As a reference, the Banana Pi M3 specifications can be referred to in [26].

The drawback of this SBC, related to the facial recognition program case, is the storage media bottleneck. This problem causes the time to retrieve image data to be very slow.

3.2. **Face dataset.** The face dataset used in this work originated from AT&T [27], which formerly originated from Olivetti Research Laboratory [28]. There are 400 pictures for 40 people, with ten images per person. All images have slight variations in expression, poses, and varying degrees of intensity. Among the ten images for each person, nine are used for training, and one is used for inference tests. An example of an image from this face database can be seen in Figure 1.

4. Results and Discussion.

4.1. **The Eigenface number for face recognition.** In this experiment, the program was run in serial with different parameter Eigenface from one to many numbers of people's faces, namely 40. Then the value of 28 is obtained for the Eigenface variable with a maximum level of recognition rate. The difference in the level of accuracy of each Eigenface value can be seen in Figure 2. The next experiment with the Eigenface value is 28.

FIGURE 1. AT&T face database subset

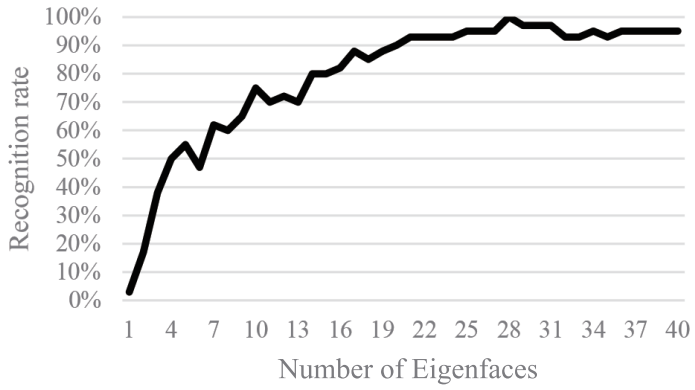

FIGURE 2. The number of Eigenfaces and recognition rate. The best number of Eigenface is 28 where the recognition rate is 100%.

4.2. **Serial program execution profile.** In this experiment, the program ran sequentially and measured the execution time of various parts of the program, such as reading the facial image training data, calculating the mean image, subtracting the mean image, transposing, calculating the covariance matrix, calculating the eigenvalue, calculating the Eigenfaces, and calculating the weights. The experiment was conducted ten times, and the average execution time is presented as the execution profile in Figure 3.

Four tasks of the Eigenface algorithm have a high execution time: reading training data, covariance matrix, calculating Eigenfaces and calculating the weight. The images are being read in the read training data section, so this process takes much time. At the same time, the other three parts involve a heavy computational process which will be optimized in the following experiment.

4.3. **Program enhancement by multithreading.** Figure 4 shows the experimental results. The results show that the larger the image size, the longer the execution time. In addition, the more threads used, the shorter the execution time. The shortest execution time of the same image size is the best. Hence, multithreading programming applied to three tasks of the Eigenface algorithm exhibits improvement.

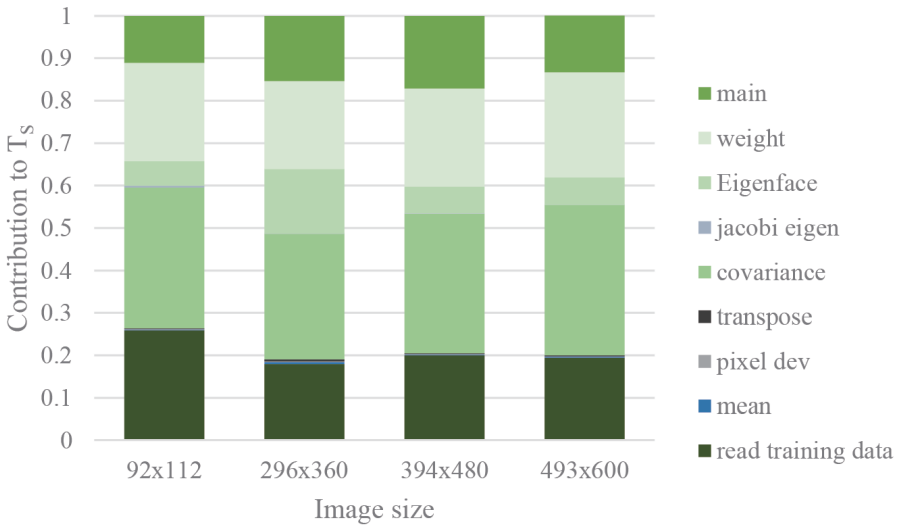

FIGURE 3. (color online) Execution time profile. Calculating the covariance matrix, Eigenface and weight spend time alot. Reading data training function is serial.

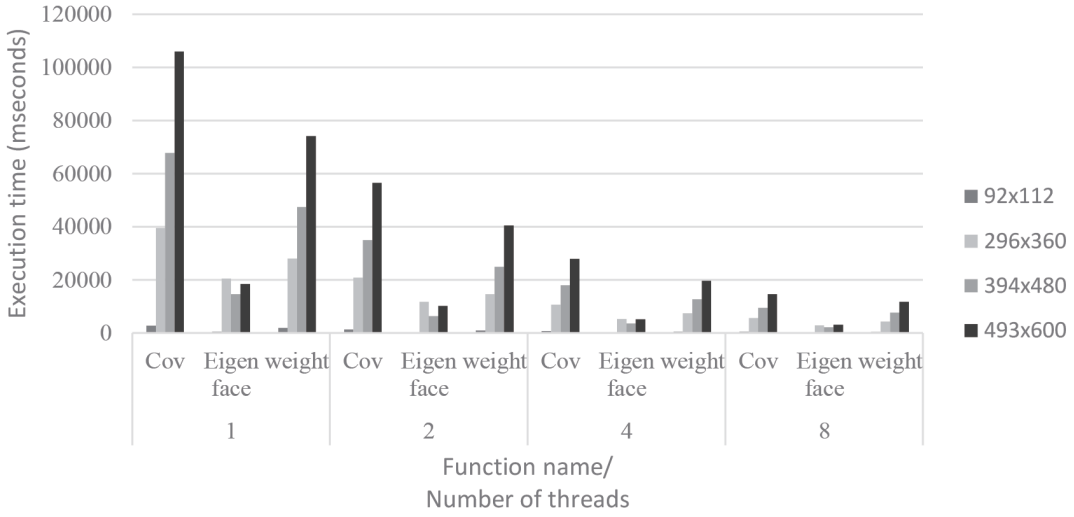

FIGURE 4. Execution time of three serial and enhanced tasks. The larger of image size, the longer the execution time. The more the threads, the shorter the execution time.

Fractional speedup analysis in Figure 5 describes how effective and efficient the parallel task is. The three enhanced tasks have good speedup values using eight threads. Improvement for the covariance matrix reaches 7.53 times fold for the 394×480 image size. This result is the most efficient since the CPU utilization score is 7.53 out of 8. That means an efficiency of 80%. Improvement for calculating Eigenface reaches 7.4 times fold for the 296×360 image size. Furthermore, an improvement over the calculating weights task of the algorithm reached 6.5 times on 296×360 image size.

4.4. **Critical path and overall speedup analysis.** In this section, we analyze the critical path and overall speedup. This overall speedup analysis involves all fractions, namely serial fractions and parallel fractions. The dominant serial fractions are the main function and the task of reading image data from storage media. The parallel fraction consists of three sub-fractions, namely the task of calculating the covariance matrix, the task of calculating the Eigenface, and calculating projections to eigenspace. In this

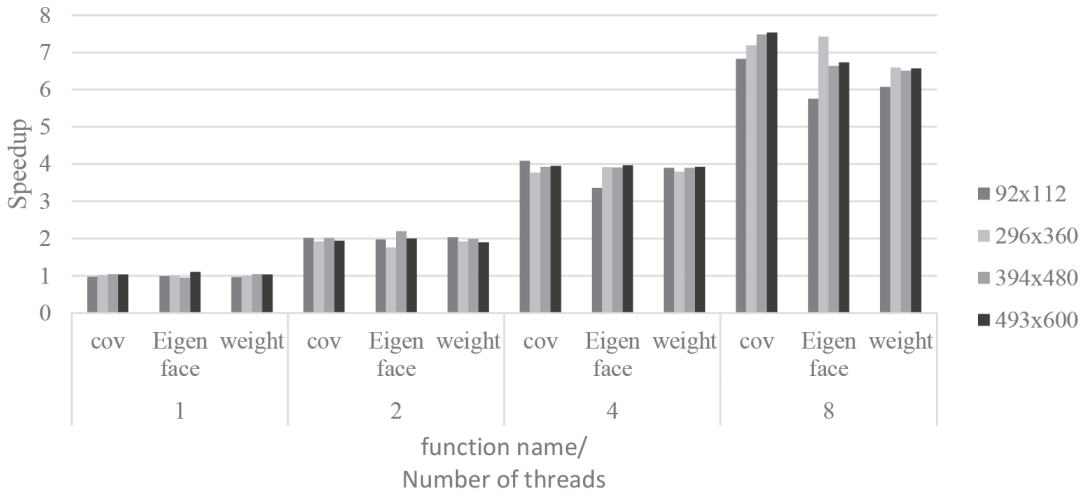

FIGURE 5. Speedup of three enhanced functions by multithreading. The more the number of threads, the higher the speedup.

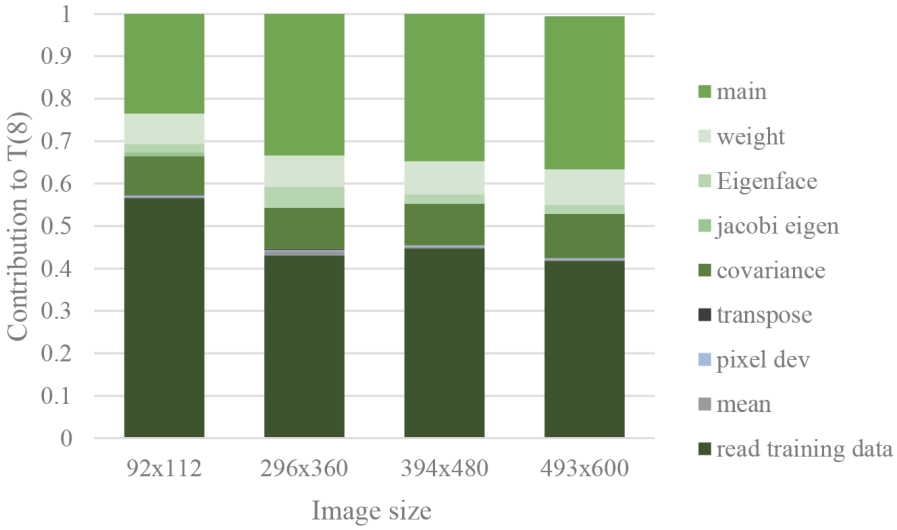

FIGURE 6. (color online) Serial and parallel fractions contributions to the critical path. Parallelization on three tasks reduced their contribution to the critical path.

case, the serial fraction and its three parallel sub-fractions establish the longest execution path, called the critical path. The composition and portion of each of these fractions are presented in Figure 6. The results were obtained from the execution of eight threads. The results show that total serial fractions contribute most to the critical path ($T(8)$). Specifically, reading the dataset contributes between 41% to 57%, while the main function contributes between 23% to 36% of the total $T(8)$.

In cases where the serial fraction contributes the most to the critical path, the overall speedup of parallel programs is bound to be as low as 2.35 times, as presented in Figure 7, even though there are eight total threads. This is equivalent to 30% efficiency.

In the case Eigenface program on Banana Pi M3, the cause of the high serial fraction is the operation of accessing image media with very low IO bandwidth. This problem can be overcome by replacing micro-SD with faster storage media. There is a main function which also contributes to the critical path; perhaps it can be overcome by extracting more parallelism from that function.

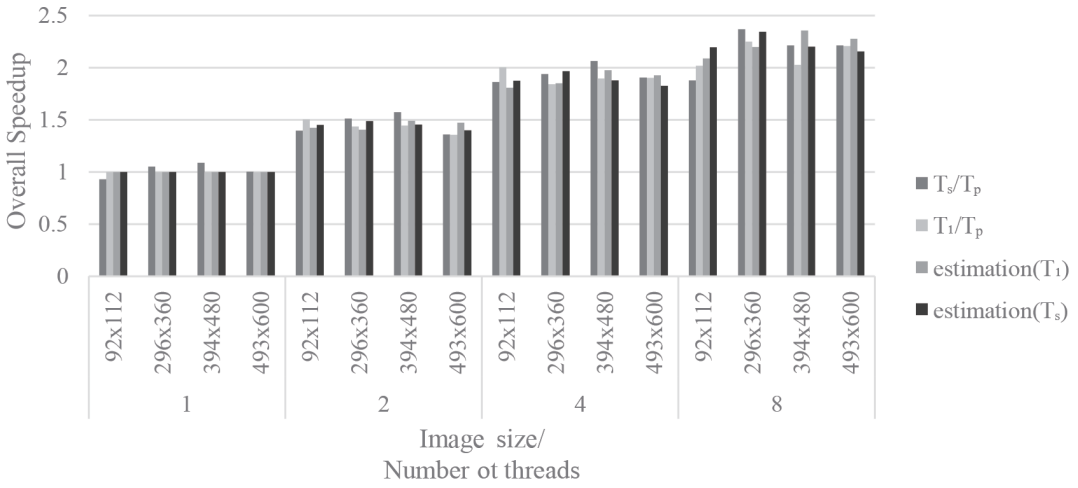

FIGURE 7. Overall speedup of the multithreaded Eigenface program. The more the threads, the higher the speedup.

5. **Conclusions.** This study proposes a multithreading programming method to improve the performance of the Eigenface program. Multithreading is applied to the three functions of the program that have long execution times. The performance of the three parallelized functions demonstrates a significant improvement compared to the serial versions. Specifically, the highest speedup achieved for the three functions, namely covariance matrix, Eigenface, and weight, on eight-core Banana Pi M3, are 7.53, 7.4, and 6.5, respectively.

The overall speedup is limited by the significant amount of serial fractions in the Eigenface program. The estimated speedup based on Amdahl's law, using both T_1 and T_S, shows that the highest speedup achieved is around 2.35 times.

The constituent functions of the serial fraction that contribute the most to $T(8)$ are reading datasets ranging from 41% to 57% and the main function ranging from 23% to 36%. This indicates that there is still room for improvement to overcome the problem of limited storage media bandwidth.

REFERENCES

[1] S. P. VanderWiel, D. Nathanson and D. J. Lilja, Complexity and performance in parallel programming languages, *Proceedings 2nd International Workshop on High-Level Parallel Programming Models and Supportive Environments*, pp.3-12, 1997.

[2] A. Goel and K. Munagala, Complexity measures for Map-Reduce, and comparison to parallel computing, *CoRR*, http://arxiv.org/abs/1211.6526, 2012.

[3] R. Bagrodia, R. Meyer, M. Takai, Y.-A. Chen, X. Zeng, J. Martin and H. Y. Song, Parsec: A parallel simulation environment for complex systems, *Computer*, vol.31, no.10, pp.77-85, 1998.

[4] H. Khaleghzadeh, H. Deldari, R. Reddy and A. Lastovetsky, Hierarchical multicore thread mapping via estimation of remote communication, *The Journal of Supercomputing*, vol.74, pp.1321-1340, 2018.

[5] J. Lee, H. Ko, J. Kim and S. Pack, DATA: Dependency-aware task allocation scheme in distributed edge clouds, *IEEE Transactions on Industrial Informatics*, vol.16, no.12, pp.7782-7790, 2020.

[6] R. Hoque, T. Herault, G. Bosilca and J. Dongarra, Dynamic task discovery in PaRSEC: A dataflow task-based runtime, *Proc. of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, New York, NY, USA, DOI: 10.1145/3148226.3148233, 2017.

[7] D. C. Brock and G. E. Moore, *Understanding Moore's Law: Four Decades of Innovation*, Chemical Heritage Foundation, 2006.

[8] D. Geer, Chip makers turn to multicore processors, *Computer*, vol.38, no.5, pp.11-13, 2005.

[9] B. A. Nayfeh and K. Olukotun, A single-chip multiprocessor, *Computer*, vol.30, no.9, pp.79-85, 1997.

[10] L. He, Computer architecture education in multicore era: Is the time to change, *2010 3rd International Conference on Computer Science and Information Technology*, vol.9, pp.724-728, 2010.

[11] K. Olukotun and L. Hammond, The future of microprocessors: Chip multiprocessors' promise of huge performance gains is now a reality, *Queue*, vol.3, no.7, pp.26-29, DOI: 10.1145/1095408.1095418, 2005.

[12] P. F. Gorder, Multicore processors for science and engineering, *Computing in Science & Engineering*, vol.9, no.2, pp.3-7, 2007.
[13] Adnan, A. Achmad and L. R. Emakarim, Leveraging ACPI _PSS data to estimate energy of processor core between ACPI-cpufreq and Intel P-State driver on low-end Intel Pentium Celeron N2830, *International Journal of Innovative Computing, Information and Control*, vol.13, no.6, pp.1993-2008, 2017.
[14] C. Tseng and S. Figueira, An analysis of the energy efficiency of multi-threading on multi-core machines, *International Conference on Green Computing*, pp.283-290, 2010.
[15] E. Antolak and A. Pułka, Energy-efficient task scheduling in design of multithread time predictable real-time systems, *IEEE Access*, vol.9, pp.121111-121127, 2021.
[16] Adnan, I. S. Areni, M. Iqbal and Y. Andyani, Smart laboratory system using Raspberry Pi 2, *ICIC Express Letters, Part B: Applications*, vol.8, no.4, pp.763-766, 2017.
[17] A. Imae, O. Koyama, I. Tomo, M. Yamaguchi, K. Ikeda and M. Yamada, Low-cost network control system based on software-defined networking over world wide web using single-board computer, *International Journal of Innovative Computing, Information and Control*, vol.18, no.3, pp.755-767, 2022.
[18] Adnan, I. S. Areni, Z. Tahir, Bayazid and Riswandi, Feasibility study on the use of low-power mini PC for magnetic resonance imaging application, *AIP Conference Proceedings*, vol.2543, no.1, DOI: 10.1063/5.0094730, 2022.
[19] K. Ogura, N. Shigei and H. Miyajima, Robust person tracking by pan-tilt camera with low-cost single board computer, *ICIC Express Letters*, vol.16, no.4, pp.381-389, 2022.
[20] G. M. Zafaruddin and H. S. Fadewar, Face recognition using eigenfaces, in *Computing, Communication and Signal Processing*, B. Iyer, S. L. Nalbalwar and N. P. Pathak (eds.), Singapore, Springer Singapore, 2019.
[21] D. Schofield, A. Nagrani, A. Zisserman, M. Hayashi, T. Matsuzawa, D. Biro and S. Carvalho, Chimpanzee face recognition from videos in the wild using deep learning, *Science Advances*, vol.5, no.9, https://www.science.org/doi/abs/10.1126/sciadv.aaw0736, 2019.
[22] S. Sapna, R. Anjali and S. N. Kamath, Performance analysis of parallel implementation of PCA-based face recognition using OpenCL, *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, pp.877-881, 2019.
[23] O. Adreeva, *EigenFace*, https://github.com/olya-d/eigenface, 2014.
[24] T. Lindner, D. Wyrwał, M. Białek and P. Nowak, Face recognition system based on a single-board computer, *2020 International Conference Mechatronic Systems and Materials (MSM)*, pp.1-6, 2020.
[25] G. M. Amdahl, Computer architecture and Amdahl's law, *Computer*, vol.46, no.12, pp.38-46, 2013.
[26] Sinovoip, *Banana Pi BPI-M3*, https://wiki.banana-pi.org/Banana_Pi_BPI-M3, 2022.
[27] AT&T Laboratories Cambridge, *The Database of Faces*, https://cam-orl.co.uk/facedatabase.html, 2001.
[28] F. S. Samaria and A. C. Harter, Parameterisation of a stochastic model for human face identification, *Proc. of 1994 IEEE Workshop on Applications of Computer Vision*, pp.138-142, 1994.